

Program pHpred.c

```
/* This software was developed at the National Institute of */
/* Standards and Technology by employees of the Federal Government */
/* in the course of their official duties. Pursuant to title 17 */
/* Section 105 of the United States Code this software is not */
/* subject to copyright protection and is in the public domain. */
/* CEMHYD3D is an experimental system. NIST assumes no */
/* responsibility whatsoever for its use by other parties, and */
/* makes no guarantees, expressed or implied, about its quality, */
/* reliability, or any other characteristic. We would appreciate */
/* acknowledgement if the software is used. This software can be */
/* redistributed and/or modified freely provided that any */
/* derivative works bear some notice that they are derived from it, */
/* and any modified versions bear some notice that they have been */
/* modified. */

/* Note that everything is being done on a one gram cement basis */
/* and we are assuming that 1 pixel is equivalent to 1 micrometer */
#define VOLFACTOR 0.00001 /* dm per pixel Note- dm*dm*dm = Liters */
#define MASSFACTOR 0.0001 /* cm per pixel - specific gravities in g/cm^3 */
/* Molar masses of ions and oxides from sodium and potassium */
/* Na = sodium, K = potassium */
#define MMNa 22.9898
#define MMK 39.102
#define MMNa20 61.979
#define MMK20 94.203
/* Basis for B factors must be adapted from 100 g to 1 g */
/* Reference: Taylor, H.F.W., "A Method for Predicting Alkali Ion */
/* Concentrations in Cement Pore Solutions," Advances in Cement Research */
/* Vol. 1, No. 1, 5-16, 1987. */
#define BNa 0.00031 /* From Taylor paper in liters (31 mL/1000/ 100 g) */
#define BK 0.00020 /* From Taylor paper in liters (20 mL/1000/ 100 g) */
#define BprimeNa 0.0030 /* From Taylor paper in liters (3 mL/1000/ 1 g POZZ) */
#define BprimeK 0.0033 /* From Taylor paper in liters (3.3 mL/1000/ 1 g POZZ) */
/* Ksp values for CH and gypsum from Reardon*/
#define KspCH25C 0.00000646
#define KspGypsum 0.0000263
/* Approximate Ksp value for syngenite from Gartner, Tang, and Weiss */
/* JACerS, Vol. 68 (12), 667-673, 1985. */
#define KspSyngenite 0.00000010
#define SpecgravSyngenite 2.607 /* Source Taylor, H.F.W., Cement Chemistry */
#define KperSyn 2.0 /* moles of K+ per mole of syngenite */
/* Some activity stuff */
#define activeA0 0.0366 /* A at 295 K (from Ken Snyder) */
#define activeB0 0.01035 /* B at 295 K (from Ken Snyder) */
/* z are the absolute charges (valences) per ion */
#define zCa 2.
#define zSO4 2.
#define zOH 1.
#define zNa 1.
#define zK 1.
/* a is similar to an ionic radius (in Angstroms) */
#define aK 1.33
#define aCa 1.
```

```

#define aOH 3.
#define aNa 3.
#define aSO4 4.5      /* Estimate as S ionic radii + O ionic diameter */
/* Ionic conductivities (From Snyder, Feng, Keen, and Mason) */
/* and from CRC Hanbook of Chemistry and Physics (1983) pp. D-175 */
/* pore solution conductivity = sum (zi * [i]*lambdai) */
/* lambdai = (lambdai_0/(1.+Gi*(Istrength^0.5))) */
/* where Istrength is in units of M (mol/L) */
#define lambdaOH_0 198.0      /* Units: S cm-cm eq.^(-1) */
#define lambdaNa_0 50.1
#define lambdaK_0 73.5
#define lambdaSO4_0 39.5
#define lambdaCa_0 29.5      /* Note that CRC has 60./2 for this */
#define GOH 0.353            /* Units: (eq.^2 mol/L)^(-0.5) */
#define GK 0.548
#define GNa 0.733
#define GCa 0.771
#define GSO4 0.877
#define cm2perL2m 0.1        /* Conversion from cm2/Liter to 1/m */

/* From Numerical Recipes in C by Press et al. */

#include "nrutil.c"
#include "complex.c"

#define EPSS 6.e-8
#define MAXIT 100

/* From Numerical Recipes in C by Press et al. */

void laguer(a,m,x,eps,polish)
fcomplex a[],*x;
int m,polish;
float eps;
{
    int j,iter;
    float err,dxold,cdx,abx;
    fcomplex sq,h,gp,gm,g2,g,b,d,dx,f,x1;
    void nrerror();

    dxold=Abs(*x);
    for (iter=1;iter<=MAXIT;iter++) {
        b=a[m];
        err=Abs(b);
        d=f=Complex(0.0,0.0);
        abx=Abs(*x);
        for (j=m-1;j>=0;j--) {
            f=Cadd(Cmul(*x,f),d);
            d=Cadd(Cmul(*x,d),b);
            b=Cadd(Cmul(*x,b),a[j]);
            err=Abs(b)+abx*err;
        }
        err *= EPSS;
        if (Abs(b) <= err) return;
        g=Cdiv(d,b);
        g2=Cmul(g,g);
        h=Csub(g2,RCmul(2.0,Cdiv(f,b)));
    }
}

```

```

        sq=Csqrt(RCmul((float) (m-1),Csub(RCmul((float) m,h),g2)));
        gp=Cadd(g,sq);
        gm=Csub(g,sq);
        if (Cabs(gp) < Cabs(gm)) gp=gm;
        dx=Cdiv(Complex((float) m,0.0),gp);
        x1=Csub(*x,dx);
        if (x->r == x1.r && x->i == x1.i) return;
        *x=x1;
        cdx=Cabs(dx);
        if (iter > 6 && cdx >= dxold) return;
        dxold=cdx;
        if (!polish)
            if (cdx <= eps*Cabs(*x)) return;
    }
    nrerror("Too many iterations in routine LAGUER");
}

#define EPSS
#define MAXIT

#define EPS 2.0e-6
#define MAXM 100

/* From Numerical Recipes in C by Press et al. */

void zroots(a,m,roots,polish)
fcomplex a[],roots[];
int m,polish;
{
    int jj,j,i;
    fcomplex x,b,c,ad[MAXM];
    void laguer();

    for (j=0;j<=m;j++) ad[j]=a[j];
    for (j=m;j>=1;j--) {
        x=Complex(0.0,0.0);
        laguer(ad,j,&x,EPS,0);
        if (fabs(x.i) <= (2.0*EPS*fabs(x.r))) x.i=0.0;
        roots[j]=x;
        b=ad[j];
        for (jj=j-1;jj>=0;jj--) {
            c=ad[jj];
            ad[jj]=b;
            b=Cadd(Cmul(x,b),c);
        }
    }
    if (polish)
        for (j=1;j<=m;j++)
            laguer(a,m,&roots[j],EPS,1);
    for (j=2;j<=m;j++) {
        x=roots[j];
        for (i=j-1;i>=1;i--) {
            if (roots[i].r <= x.r) break;
            roots[i+1]=roots[i];
        }
    }
}

```

```

        roots[i+1]=x;
    }
}

#undef EPS
#undef MAXM

void pHpred(){
    int j,syngen_change=0,syn_old=0;
    double concnaplus,conckplus;
    double concohminus,A,B,C,conctest,concsulfate1;
    double volpore,grams_cement;
    double releasedna,releasedk,activitySO4,activityK,test_precip;
    double activityCa,activityOH,Istrength,Anow,Bnow,Inew;
    double lambdaasum=0.0,conductivity=0.0;
    fcomplex coef[5],roots[5];
    float sumbest,sumtest,pozzreact,KspCH;

    /* Update CH activity product based on current system temperature */
    /* Factors derived from fitting CH solubility vs. temperature */
    /* data in Taylor book (p. 117) */
    KspCH=KspCH25C*(1.534385-0.02057*temp_cur);

    if(conccaplus>1.0){conccaplus=0.0;}
    /* Calculate volume of pore solution in the concrete in Liters */

volpore=(double)count[POROSITY]+0.38*(double)count[CSH]+0.20*(double)count[POZZCSH]
+0.20*count[SLAGCSH];
    /* Convert from pixels (cubic micrometers) to liters (cubic decimeters) */
    volpore*=VOLFACTOR;
    volpore*=VOLFACTOR;
    volpore*=VOLFACTOR;
    /* Compute pore volume per gram of cement */
    grams_cement=cemmasswgyp*MASSFACTOR*MASSFACTOR*MASSFACTOR;
    /* Compute pore volume per gram of cement */
    volpore/=grams_cement;
    /* Compute grams of pozzolan which have reacted */
    pozzreact=((float)npr/1.35)*MASSFACTOR*MASSFACTOR*MASSFACTOR*specgrav[POZZ];
    /* Compute moles of released potassium and sodium per gram of cement*/
    if(time_cur>1.0){
        releasedk=(2.)*(rspotassium+(totpotassium-rspotassium)*alpha_cur));
        releasedk/=MMK2O;
        releasedna=(2.)*(rssodium+(totsodium-rssodium)*alpha_cur));
        releasedna/=MMNa2O;
    }
    else{
        /* Proportion the early time release over the first hour */
        /* 90% immediately and the remaining 10% over the first hour */
        /* based on limited data from Davide Zampini (MBT) */
        releasedk=(2.*((0.9+0.1*time_cur)*(rspotassium)+(totpotassium-
rspotassium)*alpha_cur));
        releasedk/=MMK2O;
        releasedna=(2.*((0.9+0.1*time_cur)*(rssodium)+(totsodium-
rssodium)*alpha_cur));
        releasedna/=MMNa2O;
    }
    /* Compute concentrations of K+ and Na+ in pore solution currently */
}

```

```

/* Remember to decrease K+ by KperSyn*moles of syngenite precipitated */
/* Units must be in moles/gram for both */
conckplus=(releasedk-
moles_syn_precip*KperSyn)/(volpore+BK*alpha_cur+BprimeK*pozzreact));
concnaplus=(releasedna)/(volpore+BNa*alpha_cur+BprimeNa*pozzreact);

do{ /* while Syngenite precipitating loop */
/* Now compute the activities (estimated) of Ca++ and OH- */
activityCa=activityOH=activitySO4=activityK=1.0;
Inew=0.0;
if(((concnaplus+conckplus)>0.0)&&(soluble[ETTR]==0)){
    /* Factor of 1000 to convert from M to mmol/L */
    Istrength=1000.*(zK*zK*conckplus+zNa*zNa*concnaplus+zCa*zCa*conccaplus);
    if(Istrength<1.0){Istrength=1.0;}
    while((abs(Istrength-Inew)/Istrength)>0.10){

        Istrength=1000.*(zK*zK*conckplus+zNa*zNa*concnaplus+zCa*zCa*conccaplus);
        if(Istrength<1.0){Istrength=1.0;}

Anow=activeA0*295.*sqrt(295.)/((temp_cur+273.15)*sqrt(temp_cur+273.15));
Bnow=activeB0*sqrt(295.)/(sqrt(temp_cur+273.15));
/* Equations from papers of Marchand et al. */
activityCa=(-
Anow*zCa*zCa*sqrt(Istrength))/(1.+aCa*Bnow*sqrt(Istrength));
activityCa+=(0.2-
0.0000417*Istrength)*Anow*zCa*zCa*Istrength/sqrt(1000.);
activityCa=exp(activityCa);
activityOH=(-Anow*zOH*zOH*sqrt(Istrength))/(
1.+aOH*Bnow*sqrt(Istrength));
activityOH+=(0.2-0.0000417*Istrength)
*Anow*zOH*zOH*Istrength/sqrt(1000.);
activityOH=exp(activityOH);
activityK=(-Anow*zK*zK*sqrt(Istrength))/(1.+aK*Bnow*sqrt(Istrength));
activityK+=(0.2-0.0000417*Istrength)*Anow*
zK*zK*Istrength/sqrt(1000.);
activityK=exp(activityK);
activitySO4=(-Anow*zSO4*zSO4*sqrt(Istrength))/(
1.+aSO4*Bnow*sqrt(Istrength));
activitySO4+=(0.2-0.0000417*Istrength)*Anow*zSO4*
zSO4*Istrength/sqrt(1000.);
activitySO4=exp(activitySO4);
/* Now try to find roots of fourth degree polynomial */
/* to determine sulfate, OH-, and calcium ion concentrations */
/* A=(-KspCH); */
/* Now with activities */
A=(-KspCH/(activityCa*activityOH*activityOH));
B=conckplus+concnaplus;
C=(-2.*KspGypsum/(activityCa*activitySO4));
concohminus=conckplus+concnaplus;
coef[0]=Complex(C,0.0);
coef[1]=Complex((A+2.*B*C)/C,0.0);
coef[2]=Complex(B*B/C+4.,0.0);
coef[3]=Complex(4.*B/C,0.0);
coef[4]=Complex(4./C,0.0);
/*
printf("coef 0 is (%f,%f)\n",coef[0].r,coef[0].i);
printf("coef 1 is (%f,%f)\n",coef[1].r,coef[1].i);
printf("coef 2 is (%f,%f)\n",coef[2].r,coef[2].i);

```

```

        printf("coef 3 is (%f,%f)\n",coef[3].r,coef[3].i);
        printf("coef 4 is (%f,%f)\n",coef[4].r,coef[4].i); */
roots[1]=Complex(0.0,0.0);
roots[2]=Complex(0.0,0.0);
roots[3]=Complex(0.0,0.0);
roots[4]=Complex(0.0,0.0);
zroots(coef,4,roots,1);
sumbest=100;
/* Find the best real root for electoneutrality */
for(j=1;j<=4;j++){
    printf("pH root %d is (%f,%f)\n",j,roots[j].r,roots[j].i);
    fflush(stdout);
    if(((roots[j].i)==0.0)&&((roots[j].r)>0.0)){
conctest=sqrt(KspCH/(roots[j].r*activityCa*activityOH*activityOH));
concsulfate1=KspGypsum/(roots[j].r*activityCa*activitySO4);
sumtest=concnaplus+conckplus+2.*roots[j].r-conctest-2.*concsulfate1;
    if(fabs(sumtest)<sumbest){
        sumbest=fabs(sumtest);
        concohminus=conctest;
        concapplus=roots[j].r;
        concsulfate=concsulfate1;
    }
}
/* Update ionic strength */
Inew=1000.*(zK*zK*conckplus+zNa*zNa*concnaplus+zCa*zCa*conccapplus);
} /* end of while loop for Istrength-Inew */
}
else{
/* Factor of 1000 to convert from M to mmol/L */
Istrength=1000.*(zK*zK*conckplus+zNa*zNa*concnaplus+zCa*zCa*conccapplus);
if(Istrength<1.0){Istrength=1.0;}
while((abs(Istrength-Inew)/Istrength)>0.10){

Istrength=1000.*(zK*zK*conckplus+zNa*zNa*concnaplus+zCa*zCa*conccapplus);
Anow=activeA0*295.*sqrt(295.)/((temp_cur+273.15)*sqrt(temp_cur+273.15));
Bnow=activeB0*sqrt(295.)/(sqrt(temp_cur+273.15));
/* Equations from papers of Marchand et al. */
activityCa=(-Anow*zCa*zCa*sqrt(Istrength))/(
    (1.+aCa*Bnow*sqrt(Istrength)));
activityCa+=(0.2-0.0000417*Istrength)*Anow*zCa*(
    zCa*Istrength/sqrt(1000.));
activityCa=exp(activityCa);
activityOH=(-Anow*zOH*zOH*sqrt(Istrength))/(
    (1.+aOH*Bnow*sqrt(Istrength)));
activityOH+=(0.2-0.0000417*Istrength)*Anow*zOH*(
    zOH*Istrength/sqrt(1000.));
activityOH=exp(activityOH);
activityK=(-Anow*zK*zK*sqrt(Istrength))/(1.+aK*Bnow*sqrt(Istrength));
activityK+=(0.2-0.0000417*Istrength)*Anow*zK*(
    zK*Istrength/sqrt(1000.));
activityK=exp(activityK);
/* Calculate pH assuming simply that OH- balances sum of Na+ and K+ */
concohminus=conckplus+concnaplus;
if((concnaplus)>(0.1*(concohminus))){
    concohminus+=(2.*concnaplus);
}
}

```

```

conccapplus=(KspCH/(activityCa*activityOH*activityOH*concohminus*concohminus));
concsulfate=0.0;
/* Update ionic strength */
Inew=1000.* (zK*zK*conckplus+zNa*zNa*concnaplus+zCa*zCa*conccapplus);
} /* end of while loop for Istrength-Inew */
}
/* Check for syngenite precipitation */
syngen_change=0;
if(syn_old!=2){
    test_precip=conckplus*conckplus*activityK*activityK;
    test_precip*=conccapplus*activityCa;
    test_precip*=concsulfate*concsulfate*activitySO4*activitySO4;
    if(test_precip>KspSyngenite){
        printf("Syngenite precipitating at cycle %d\n",icyc);
        syngen_change=syn_old=1;
        /* Units of moles_syn_precip are moles per gram of cement */
        if(conckplus>0.002){
            conckplus-=0.001;
            moles_syn_precip+=0.001*volpore/KperSyn;
        }
        else if(conckplus>0.0002){
            conckplus-=0.0001;
            moles_syn_precip+=0.0001*volpore/KperSyn;
        }
        else{
            moles_syn_precip+=conckplus*volpore/KperSyn;
            conckplus=0.0;
        }
    }
    /* Check for syngenite dissolution */
    /* How to control dissolution rates??? */
    /* Have 0.001*KperSyn increase in conckplus each cycle */
    /* Only one dissolution per cycle --- purpose of syn_old */
    /* and no dissolution if some precipitation in that cycle */
    if((syn_old==0)&&(moles_syn_precip>0.0)){
        syngen_change=syn_old=2;
        conckplus+=(moles_syn_precip/10.0)/volpore; */
        if((moles_syn_precip/volpore)>0.001){
            conckplus+=0.001*KperSyn;
            moles_syn_precip-=(0.001*volpore);
        }
        else{
            conckplus+=(moles_syn_precip*KperSyn/volpore);
            moles_syn_precip=0.0;
        }
    }
}
} while(syngen_change!=0);

if(concohminus<(0.0000001)){
    concohminus=0.0000001;
conccapplus=(KspCH/(activityCa*activityOH*activityOH*concohminus*concohminus));
}
pH_cur=14.0+log10(concohminus*activityOH);
/* Calculation of solution conductivity (Snyder and Feng basis) */
/* First convert ionic strength back to M units */
Istrength/=1000.;


```

```

conductivity+=zCa*conccapplus*(lambdaCa_0/(1.+GCa*sqrt(Istrength)));
conductivity+=zOH*concohminus*(lambdaOH_0/(1.+GOH*sqrt(Istrength)));
conductivity+=zNa*concnaplus*(lambdaNa_0/(1.+GNa*sqrt(Istrength)));
conductivity+=zK*conckplus*(lambdaK_0/(1.+GK*sqrt(Istrength)));
conductivity+=zSO4*concsulfate*(lambdaSO4_0/(1.+GSO4*sqrt(Istrength)));
conductivity*=cm2perL2m;

/* Output results to logging file */
pHfile=fopen(pHname,"a");

if((cycCnt-1)==0){
    fprintf(pHfile,"Cycle time(h) alpha_mass pH sigma(S/m) [Na+] [K+] [Ca++]
[SO4--] activityCa activityOH activitySO4 activityK molesSynGenite\n");
}
fprintf(pHfile,"%d %.4f %.4f
%f\n",cycCnt-1,time_cur,alpha_cur,pH_cur,conductivity,concnaplus,
conckplus,conccapplus,concsulfate,activityCa,activityOH,activitySO4,
activityK,moles_syn_precip);
fclose(pHfile);

}

```